2017 IEEE 33rd International Conference on Data Engineering

Tracking Matrix Approximation over Distributed Sliding Windows

Haida Zhang* Zengfeng Huang*
School of CSE, UNSW
{haida.zhang, zengfeng.huang}@unsw.edu.au

Zhewei Wei*
Renmin University of China
zhewei@ruc.edu.cn

Wenjie Zhang* Xuemin Lin*
School of CSE, UNSW
{zhangw, lxue}@cse.unsw.edu.au

***Abstract*—In many modern applications, input data is represented as matrices and often arrives continuously. The ability to summarize and approximate data matrices in streaming fashion has become a common requirement in many emerging environments. In these applications, input data is usually generated at multiple distributed sites and simply centralizing all data is often infeasible. Therefore, novel algorithmic techniques are required. Furthermore, in most of these applications, queries must be answered solely based on the recently observed data points (e.g., data collected over the last hour/day/month), which makes the problem even more challenging. In this paper, we propose to study the problem of tracking matrix approximations over distributed sliding windows. In this problem, there are m distributed sites each observing a stream of d-dimensional data points. The goal is to continuously track a small matrix B as an approximation to A_w, the matrix consists of data points in the union of the streams which arrived during the last W time units. The quality of the approximation is measured by the *covariance error* $\|A_w^T A_w - B^T B\|/\|A\|_F^2$ [1], and the primary goal is to minimize communication, while providing provable error guarantee. We propose novel communication-efficient algorithms for this problem. Our sampling-based algorithms continuously track a weighted sample of rows according to their squared norms, which generalize and simplify the sampling techniques in [2]. We also propose deterministic tracking algorithms that require only one-way communication and provide better error guarantee. All algorithms have provable guarantees, and extensive experimental studies on real and synthetic datasets validate our theoretical claims and demonstrate the efficiency of these algorithms.**

I. Introduction

In many applications, including network traffic monitoring, financial data analysis, and real-time anomaly detection, data is received as continuous high-volume streams. Under such settings, decisions usually have to be made in real time (e.g., detection of DDoS attacks), and thus traditional database query processing methods are not suitable for such applications. The ability to summarize massive streaming data continuously becomes crucial, i.e., dynamically maintaining a compact *summary* (a.k.a. synopsis or sketch) of the input stream, which can be used to provide *approximate* query answers.

Large-scale streaming processing applications are also inherently *distributed*: streaming tuples are continuously received at multiple, possibly geographically dispersed, distributed sites (sensor nodes, routers, smart phones, etc). Efficiently tracking the value of a function over the union of streams has become a key procedure to support real-time decision making such as detection of fraudulent transactions, user interest prediction, and network health monitoring. This motivates the *distributed monitoring model* [3], which has attracted great attention in recent years (e.g. [2, 4, 5, 6, 7]). In this model, there are multiple distributed sites, each observing a disjoint stream of items, and a single coordinator, whose goal is to monitor (or track) a function over the union of the streams *at all times*. In such distributed applications, *communication efficiency* is always a primary concern either due to limited bandwidth or other practical reasons. For example, in sensor networks, it is usually impractical to install new batteries for geographically dispersed sensor nodes, and thus power consumption is typically the main concern. As the battery drain for sending messages exceeds by several orders of magnitude the drain for local operations within a node, reducing communication is critical to sensor networks [8, 9]. Therefore, in this paper, we mainly aim at reducing the communication cost, which is also a conventional methodology used in most previous works on distributed monitoring model [1, 2, 4, 10].

Despite the success of the distributed monitoring model, a bulk of prior works focus on simple aggregate queries (e.g., counts, item frequencies, and quantiles) and low-dimensional data. In many tasks of machine learning, information retrieval, and large-scale data analytics, the input data is represented as large matrices [11, 12, 13]. For example, in textual analysis, a set of documents is modeled as a matrix whose rows correspond to different documents and columns correspond to words; in image analysis, each image is represented as a row containing either pixel values or a set of derived features. Such data is often huge, complex, and continuously generated at multiple sources, and summarizing such large matrix data has become a common requirement in many emerging application environments. Recently a new line of research, represented by [13], focuses on the *row-update streaming model*. In this model, each item in the stream is a row of a matrix; the goal is to maintain a good approximation to the matrix consisting of all the rows received so far. We have seen a flurry of activities in the area of matrix approximation on centralized streams [14, 15, 16, 17]. While as far as we know, [1] is the only work on distributed tracking of matrix approximations.

In streaming processing, items naturally carry timestamps, and recent data is usually more interesting than outdated data. For instance, in network analysis, to detect DDoS attacks, one should track frequency statistics over a short past period rather than the entire history [18]. The *sliding window model* is arguably one of the most prominent and intuitive time-decaying models [19, 20, 21], which considers only a window of the most recent items seen in the stream thus far.

*Contact authors.

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.133

Motivation. In this paper, we study the problem of continuously tracking of a matrix approximation over distributed sliding windows. We propose communication-, space- and time-efficient algorithms for maintaining a *covariance sketch*[1] over sliding windows in the distributed monitoring model. Dynamically tracking a covariance sketch has been widely used as a building block in many applications (e.g., low rank approximation [14], anomaly detection [15], online PCA [22], and spectral clustering [23]). Our problem is motivated by applications in sensor networks, distributed databases, cloud computing. We provide two concrete applications.

(1) *Approximate PCA for Change Detection.* In *principle component analysis*, the goal is to compute an orthonormal basis of a lower dimensional subspace which captures the variance of the data as much as possible. Interestingly, it was shown that the top-k right singular vectors of a *covariance sketch* approximate the optimal PCA basis of the original matrix well [14]. Therefore, we can perform PCA on a much smaller sketch matrix instead of on the original one. A particular application of PCA is to detect changes in multidimensional data streams [24]. More concretely, changes are detected by comparing the PCA basis of the *testing window* (i.e., the current window) to the PCA basis of a fixed *reference window* which has been extracted earlier. Hence, the ability to continuously maintain an (approximate) PCA basis of the current window is crucial to this approach.

(2) *Anomaly detection.* In a recent work [15], covariance sketch is used as the main tool for streaming anomaly detection in the centralized setting. Given a set of non-anomalous data points observed so far (represented as a matrix A), function $f(A, x)$ is used to measure the anomaly score of a new data point x. However it is infeasible to compute $f(A, x)$ exactly. Then it is shown that if B has small covariance error w.r.t. A, then $f(A, x)$ can be approximated accurately by $f(B, x)$, which can be computed much more efficiently in terms of running time and space. Note that in real situations, anomaly is often supposed to be detected based on data within a sliding window (due to *concept drift*, i.e., new patterns of normal data appear over time), and input data might be generated at multiple distributed sites. [15] did not address these issues, and we believe, our algorithms can be used as tools for window-based anomaly detection over distributed streams.

A. Problem definitions

Notations. We use m for the number of sites, and d for the dimension of each row. R is the maximum ratio of squared norms of any two rows in the matrix and N is the maximum number of rows in a sliding window. We remark that our protocols do not need to know R or N in advance. A_w refers to the matrix that consists of all *active* rows, i.e., rows included by current window. For a d-dimensional vector x, $\|x\|$ is the ℓ_2 norm of x. We use x_i to denote the ith entry of x. Let $A \in \mathbb{R}^{n\times d}$ be a matrix of dimension $n \times d$ with $n > d$. We use a_i to denote the ith row of A, and $a_{i,j}$ for the (i, j)-th entry of A. We write the (reduced) singular value decomposition of A as $(U, \Sigma, V) = \mathsf{SVD}(A)$. We use $\|A\|_2$ or $\|A\|$ to denote the spectral norm of A, which is the largest singular value of A, and $\|A\|_F$ for the *Frobenius Norm*, which is $\sqrt{\sum_{i,j} a_{i,j}^2}$.

[1] A type of matrix approximation that will be defined in the next section.

Notation	Descriptions
a_j	a row in the matrix stream
d	the dimension of a row
R	the maximum ratio of squared norms of rows
W	window size
A_w	the matrix within the current sliding window
N	the maximum number of rows in a sliding window
m	the number of sites
ℓ	the size of sample set
B	a matrix approximation to A_w
ε	error parameter

Table I: Symbols and descriptions

Given another matrix B with the same number of columns as A, $[A; B]$ is the matrix formed by concatenating the rows of A and B. See Table I for a summary.

Covariance sketch. In this paper, we measure the quality of matrix approximation by the well-known covariance error. More formally, given a target $n \times d$ matrix A and an approximation parameter ε, we call another matrix $B \in \mathbb{R}^{\ell\times d}$ an ε-covariance sketch of A, if the covariance error is at most ε, i.e. $\|A^T A - B^T B\|/\|A\|_F^2 \le \varepsilon$. In practice, we want $\ell \ll n$. Alternatively, it is known [13] that the covariance error can be equivalently defined as $\max_{x:\|x\|=1} |\|Ax\|^2 - \|Bx\|^2|/\|A\|_F^2$. Intuitively, a covariance sketch preserves the norm (or length) of A in any direction x.

Tracking covariance sketch over distributed time-based sliding windows. Formally, we model a data stream as an infinite sequence $S = \{(a_i, t_i) \mid i = 1, \cdots, \infty\}$, where a_i is a row (or record) with d numerical attributes and t_i is the timestamp of a_i. Assume t_{now} is the current time, given a window size W, we use A_w to denote the matrix formed by all the rows whose timestamps are in the range $(t_{now} - W, t_{now}]$, which is our target matrix. We call a row *active*, if it is in A_w. Note that the number of rows in a time window could vary drastically over time. If the window size W is infinite, the model degenerates to the standard distributed monitoring model, and we will simply call it the infinite window model. At time t_i, the record a_i appears at exactly one of the m distributed sites $S_1, S_2, \cdots, S_m$. The goal is to maintain or track a smaller matrix B which has covariance error at most ε to A_w at the coordinator site C. Each of the m sites communicates with C via a two-way communication channel.

We remark that there is an alternative sliding window model called *sequence-based sliding window*, which considers the last W items in the stream. However, as argued in [2], time-based model is often more useful in practice; besides, sequence-based model is quite different from time-based sliding window in the distributed setting (although it is a special case of time-based model in the centralized setting), thus requires a different set of techniques [2]. Therefore, attention is focused on time-based sliding window in this paper.

B. Our contributions

In this paper, we provide several algorithms/protocols for our problem. The protocols can be divided into two categories: (1) *sampling based*, and (2) *deterministic tracking*.

Sampling based. In sampling based protocols, we aim to pick random samples of rows in the sliding window with probability

Algorithms	Communication (Words)	Space per site (Words)
Priority Sampling	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR) + m\log(NR)$	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR)$
ES Sampling	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR) + \frac{m}{\varepsilon}\log(NR)$	$\frac{d}{\varepsilon^2}\log\frac{1}{\varepsilon}\log(NR)$
DA1	$\frac{md}{\varepsilon}\log(NR)$	$d^2 + \frac{d}{\varepsilon^2}\log(NR)$
DA2	$\frac{md}{\varepsilon}\log(NR)$	$\frac{d}{\varepsilon^2}\log(NR)$

Table II: Asymptotic bounds on communication and space of our protocols for tracking covariance sketch over distributed sliding windows. The cost of sampling protocols holds with high probability.

proportional to their squared norms, i.e. *weighted sampling*. Though random sampling, as a fundamental tool for analyzing big data, has been extensively studied in the centralized setting [25], it is highly non-trivial to extend it to the distributed setting. The sampling problem in the distributed sliding window model was only recently studied in [2], but so far all the techniques only work for uniform sampling. We emphasize that uniform sampling (each row is sampled with the same probability) does not work for covariance sketch. Consider an $n \times 2$ matrix A, where the first row is $[n, 0]$ and all the other rows are $[0, 1]$, then $\|A\|_F^2 = n^2 + n - 1$ and $A^T A$ is a 2×2 diagonal matrix whose upper left entry is n^2. If we uniformly sample ℓ rows, then with probability $1 - \ell/n$ (which is high when ℓ is sublinear in n) the first row will not be sampled and consequentially the covariance error of the sketch matrix is too large (at least $n^2/(n^2+n) \approx 1$ for large n). In this paper, we provide the first protocols for *continuous weighted sampling over distributed sliding windows*. In the centralized setting, there are two well-known methods for weighted sampling, namely *priority sampling* of [26] and *ES sampling* of [27]. Based on techniques from [2], we propose a general framework, which extends both methods to the distributed sliding window model. Our framework works for both sampling with and without replacement.

Deterministic tracking with one-way communication. We also provide two deterministic tracking algorithms based on two algorithmic frameworks. One simple but crucial observation is that, to track a global covariance sketch, the coordinator only needs to track each of the m local matrices separately, i.e., at any time, the coordinator maintains an ε-covariance sketch for each site separately. To do so, our first framework uses a surprisingly simple idea. Roughly speaking, each site tracks the covariance error between its local sliding-window matrix and the sketch that coordinator currently has, and will send a message only when the error exceeds a threshold. This simple idea is actually quite general, and also works for simple aggregate queries such as counting, item frequencies, and order statistics, which can simplify the algorithms in [28]. We also provide a second approach, which alternatively tracks the local matrix of each site by combining the *forward-backward* framework of [28] with *iterative matrix sketching* of [13]. One property of our deterministic protocols is they only use one-way communication, i.e., all communication is from sites to the coordinator, which is desired in many applications.

Theoretical analysis and experimental studies. We provide rigorous analysis for each of our protocols. The asymptotic bounds on communication cost and space usage are listed in Table II. Moreover, we conduct extensive experimental studies on both real and synthetic datasets to test and compare the performances of proposed algorithms.

C. Related work

Matrix approximation problems have been extensively studied in the centralized streaming model. The *row-update streaming model* is most related to our study. In this model, each item in the stream is a row of a matrix; the goal is to maintain a good approximation to the matrix consisting of all the rows received so far with limited space. Liberty [13] proposed the Frequent Direction (FD) algorithm, which maintains an ε-covariance sketch using $O(d/\varepsilon)$ space. [14, 16] improved and generalized the original FD. More recently, [17] provided algorithms for maintaining covariance sketch in the sliding window model. But none of above works can be applied in distributed setting without incurring high communication cost. Other error measures of matrix approximation are also widely studied, we refer to a recent survey by Woodruff [29].

In the popular distributed monitoring model, numerous works have been done for tracking simple aggregate queries [2, 4, 5, 6, 7]. See the recent survey by Cormode [3] for more references. A work closely related to ours is [1], which considers the problem of tracking covariance sketch in the standard distributed monitoring model. However, all of their techniques only work for unbounded streams (i.e., there is no item expiration). In the sliding window model, the algorithms not only need to process new data received, they also need to handle expirations, which is often considered the main challenge. Generally, there is no easy way to convert an unbounded streaming algorithm to a sliding window algorithm whether in the centralized or distributed setting.

Tracking simple aggregate queries, such as counts, heavy hitters, and quantiles, in the distributed sliding window model has also attracted lots of attention in recent years [2, 10, 28, 30]. However, to the best of our knowledge, there is no comprehensive study of tracking matrix approximation in this model. Cormode et. al. [2] considered the problem of tracking random samples over distributed sliding windows, but current techniques only work for unweighted sampling, and thus cannot be used to track matrix approximation.

Outline. Section II presents sampling algorithms for distributed time-based sliding window. Section III introduces deterministic algorithms to tracking sum and tracking matrix over distributed time-based sliding window, and provides firm proofs for communication complexity and space complexity of these algorithms. Section IV evaluates all the introduced algorithms using extensive experiments. Finally we conclude the paper in Section V.

II. Sampling algorithms

In the sampling based algorithms, our goal is to maintain a random sample set of rows on the coordinator, where the rows are chosen with probabilities proportional to their squared norms. The size of sample set should be at least $\ell = \Omega(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ to produce a ε-covariance sketch with high probability [1]. In the centralized setting, there are two well-known techniques for random weighted sampling, namely ES sampling [27] and priority sampling [26]. Both of them use the following framework: to sample ℓ items without replacement, the algorithm assigns each item a random priority based on its weight, and picks items with top-ℓ priorities as the sample

set. The main differences between ES sampling and priority sampling are the ways to assign priorities and create estimates.

We propose a general framework in the distributed sliding window model, which efficiently tracks the set of items with top-ℓ priorities regardless of how their priorities are calculated, and thus works for both techniques above. The key challenge is that, even when later arrivals are dominated by earlier items (in terms of priority), they cannot be simply discarded. Otherwise, when earlier items expire, there may be insufficient later items kept to provide a sample of size ℓ. This challenge also exists in unweighted sampling (due to non-uniform arrival rate of items), and was resolved in [2] either by storing all active items or using a complicated level-sampling data structure. The weighted version is more challenging, but we provide a solution, which is more general (handling weights), uses the same communication and space, and is simpler to implement. Next, we will mainly focus on priority sampling. We first show how to do sampling without replacement, then discuss sampling with replacement. ES sampling can be applied similarly. For simplicity, we assume that there are always more than ℓ active rows. Otherwise, the coordinator can retrieve all active rows.

A. Priority Sampling

Priority Sampling without Replacement (PWOR). In priority sampling, each item a_i with weight w_i is assigned a priority value $\rho_i = \frac{w_i}{u_i}$, where u_i is a random number uniformly chosen from $(0, 1)$. The goal is to track the set of items with top-ℓ priority values. In the row sampling problem, each item is a row a_i with weight $w_i = \|a_i\|^2$.

A simple protocol. Our first protocol works as follows. At any time, all the sites maintain the same threshold τ, which is the ℓ-th largest priority value among all active rows. When a new row a_i is observed by the site S_j, it is assigned a priority value ρ_i defined as above. If $\rho_i \geq \tau$, the site sends (a_i, t_i, ρ_i) to the coordinator and discards it. Otherwise, the site stores it locally until there are ℓ rows in the same site that arrive later than a_i but have higher priority values than ρ_i or until a_i is sent to the coordinator. The coordinator maintains a queue S, which contains the set of rows with top-ℓ priority values. The tuples sent from sites are inserted into S and the expired tuples ($t_i < t_{now} - W$) are removed from S. The coordinator always adjusts τ to ensure $|S| = \ell$: (1) When $|S| > \ell$, the coordinator increases τ and moves unqualified tuples ($\rho_i < \tau$) to S' (candidate set), and new τ is broadcast; (2) When $|S| < \ell$, the coordinator decreases τ by *negotiating with* all sites and S', so that the number of qualified tuples ($\rho_i \geq \tau$ and unexpired) is exactly ℓ. The new threshold τ is broadcast and all qualified tuples held by sites are forwarded to coordinator.

Our first protocol is given in **Algorithm** 1. At a site j, each observed row a_i is assigned a priority value ρ_i (line 2). The tuple (a_i, t_i, ρ_i) is sent to coordinator if ρ_i is greater than the threshold τ (line 3-4), otherwise it is appended to a local queue Q (line 5). Then site j removes all rows that expire (line 7) and rows that are dominated by ℓ later rows (line 8-11). At each timestamp, the coordinator always checks if there are new samples being received, and appends these samples to the sample set S (line 13-14). Then, the coordinator removes all rows in S and S' that expire (line 15-16). Next, the coordinator calls the procedure $Update\_threshold$, in which it adjusts τ and S if the size of sample set $|S| \neq \ell$ (line 17). Firstly, while $|S| > \ell$, the coordinator keeps moving the sample with the minimum priority value in S to the candidate sample set S' until $|S| = \ell$ (line 19-21). Notice that such rows still have the chance to become top-ℓ in term of priority when old samples expire. Secondly, if $|S| < \ell$ (due to expiration of old samples), coordinator initializes a set P by requesting from S' and each of m sites (totally $m + 1$ sources) their local highest priority values (line 22-24). Then, the algorithm repeats the following process until $|S|$ reaches ℓ: at first, the coordinator finds the highest priority ρ_p in P, and retrieves the pair (a_p, ρ_p, t_p) from the corresponding source (either a site or S', while only a retrieve from the site incurs communication), and inserts the pair into the sample set S (line 26-27); next, the coordinator removes ρ_p from P, and requests the next highest priority from the same source as that of ρ_p, i.e., either S' or a site that provided ρ_p (line 28-29). Finally, the coordinator updates τ to be the minimum priority in S, and broadcasts τ to m sites if τ has changed since the last timestamp (line 30-31).

Definition 1 (ℓ-dominance)**.** Given a sequence of rows $a_1, a_2, \cdots, a_N$, each of which is assigned a (random) priority value as in priority sampling. We say a_i is *left ℓ-dominated* if there are ℓ rows before a_i which have greater priority values than ρ_i. The *right ℓ-dominance* is defined analogously.

The correctness of the above algorithm is based on a simple observation: if a_i is right ℓ-dominated by later rows, then a_i will never be in the top-ℓ before it expires. In our protocol each site only discards rows that have been right ℓ-dominated by later rows, and thus no potential top-ℓ row is discarded. Thereby, the correctness follows, since the coordinator always maintains the global top-ℓ rows among all rows currently stored in the whole system. Despite the apparent simplicity, it's unclear whether this protocol has any theoretical guarantee on communication cost and space usage. We will give a formal analysis. The next lemma is crucial to our analysis, the proof of which is technical and is presented in our full version [31].

Lemma 1. *For a sequence of N rows with the maximum ratio of squared norms bounded by R, the number of rows that are not left ℓ-dominated is bounded by $O(\ell \log \frac{NR}{\ell})$ with probability $1 - e^{-\Omega(\ell)}$. Similarly, the number of rows that are not right ℓ-dominated is also bounded by $O(\ell \log \frac{NR}{\ell})$ with probability $1 - e^{-\Omega(\ell)}$ by symmetry.*

Lemma 2. *During a time window, with probability $1-e^{-\Omega(\ell)}$, the total communication cost is $O(d\ell \log(NR)+m\ell \log(NR))$ and each site stores $O(\ell \log(NR))$ rows, where N is the maximum number of rows in a window.*

Proof: We consider any fixed time interval $(t, t+W]$, and bound the number of rows sent in this time. Let $a_1, \cdots, a_N$ denote the sequence of rows arrive during $(t, t+W]$. Consider the rows sent immediately after being observed. By definition, it is clear that the rows sent immediately are not left ℓ-dominated in the sequence $a_1, \cdots, a_N$. According to Lemma 1, the number of such rows is bounded by $O(\ell \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$ (leads to $O(d\ell \log(NR))$ communication). After a row is sent to the coordinator, the threshold needs to be updated, which incurs $O(m)$ communication, and thus $O(m\ell \log(NR))$ words in total with probability $1 - e^{-\Omega(\ell)}$. Secondly, when a row in S expires, at most one row will be

Algorithm 1 Priority Sampling without replacement (PWOR)

1: **procedure** PROCESS_ROWS(row a_i) ▷ at site j
2: Choose $u_i \in$Unif$(0,1)$ and set $\rho_i = \frac{\|a_i\|^2}{u_i}$.
3: **if** $\rho_i \geq \tau$ **then**
4: Send (a_i, ρ_i, t_i) to coordinator.
5: **else** Append $(a_i, \rho_i, t_i, count_i = 0)$ to the end of Q.
6: **for** $(a_p, \rho_p, t_p, count_p) \in Q$ **do**
7: **if** $t_p < t - W$ **then** Remove $(a_p, \rho_p, t_p, count_p)$.
8: **if** $\rho_i \geq \rho_p$ **then**
9: $count_p += 1$.
10: **if** $count_p \geq \ell$ **then**
11: Remove $(a_p, \rho_p, t_p, count_p)$ from Q.
12: **procedure** PROCESS_SAMPLES(time t) ▷ at coordinator
13: **for** $(a_i, \rho_i, t_i) = receive\_rows(t)$ **do**
14: Insert (a_i, ρ_i, t_i) to S.
15: **for** $(a_p, \rho_p, t_p) \in S, S'$ **do**
16: **if** $t_p < t - W$ **then** Remove (a_p, ρ_p, t_p).
17: Update_threshold().
18: **procedure** UPDATE_THRESHOLD() ▷ at coordinator
19: **while** $|S| > \ell$ **do**
20: Find the minimum ρ_p in S;
21: Move (a_p, ρ_p, t_p) to candidate sample set S'.
22: **if** $|S| < \ell$ **then**
23: Initialize $P = \emptyset$;
24: Request from S' and each site their local highest priority values, and insert them into P.
25: **while** $|S| < \ell$ **do**
26: Find the highest priority ρ_p in P;
27: Retrieve (a_p, ρ_p, t_p) and insert it to S;
28: Remove ρ_p from P;
29: Request the next highest priority from the same source that provided ρ_p, and insert it into P.
30: Update τ to be the minimum priority in S;
31: Broadcast τ if τ has changed since the last timestamp.

sent to the coordinator. Note that all rows in S that expire during $(t, t+W]$ arrived in the time interval $(t-W, t]$, and the number of such rows, as shown above, is bounded by $O(\ell \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$. Therefore, the number of rows sent due to expiration of earlier samples is at most $O(\ell \log(NR))$ per window with probability $1 - e^{-\Omega(\ell)}$. When a row in S expires, $O(m)$ communication is needed to locate the row with highest priority among all unsent rows, thus $O(m\ell \log(NR))$ words in total with probability $1 - e^{-\Omega(\ell)}$. We next bound the space usage at any fixed time t'. Let $\Delta = b_1, \cdots, b_M$ be the sequence of rows observed in time interval $(t'-W, t']$ at site S_j. At time t', site S_j only maintains all rows that are not right ℓ-dominated in the sequence Δ. By Lemma 1, the number of rows stored by site S_j is bounded by $O(\ell \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$. ■

Drawbacks of Algorithm 1. In the above protocol, the threshold changes $O(\ell \log(NR))$ times per window with high probability. Each time it incurs additional $O(m)$ communication, leading to a $O(m\ell \log(NR))$ term in overall communication cost, which is significant when d is small. Moreover, every time τ changes, the whole system needs to be synchronized. In practice, frequent synchronizations greatly downgrade the overall performance of distributed systems. Motivated by these, we propose a lazy-broadcast protocol that significantly reduces the number of updates in τ.

Algorithm 2 Lazy-broadcast protocol

1: **procedure** UPDATE_THRESHOLD() ▷ at coordinator
2: **if** $|S| \geq 4\ell$ **then**
3: Assign τ the 2ℓ-th priority value in S;
4: Broadcast τ to m sites.
5: **for** each (a_p, ρ_p, t_p) in S with $\rho_p < \tau$ **do**
6: Move (a_p, ρ_p, t_p) to S'.
7: **if** $|S| \leq \ell$ **then**
8: **while** $|S| \leq 2\ell$ **do**
9: $\tau = \tau/2$;
10: Broadcast τ to m sites;
11: Collect rows from S' and all sites that satisfies $\rho > \tau$, and insert them into S.
12: **procedure** UPDATE_THRESHOLD(threshold τ) ▷ at site j
13: **if** $\tau < \tau_j$ **then**
14: **for** $(a_p, \rho_p, t_p, count_p) \in Q$ with $\rho_p \geq \tau$ **do**
15: Send (a_p, ρ_p, t_p) to coordinator;
16: Remove $(a_p, \rho_p, t_p, count_p)$ from Q.
17: Set $\tau_j = \tau$.

An improved protocol: Lazy-broadcast protocol. In lazy-broadcast protocol, the number of samples maintained by the coordinator is between ℓ and 4ℓ, i.e. $\ell \leq |S| \leq 4\ell$, rather than exactly ℓ. The coordinator only changes the threshold when the condition is violated. More precisely, when $|S|$ exceeds 4ℓ, the coordinator increases τ to remove half of rows in S; when $|S|$ is less than ℓ, the coordinator halves τ to collect more rows from sites until $|S| \geq 2\ell$.

Lazy-broadcast protocol is demonstrated in **Algorithm** 2. The procedure of processing new observed rows in each site and the procedure of processing new samples in coordinator are the same as in **Algorithm** 1, thus omitted. After processing new samples, if the coordinator finds that $|S|$ exceeds 4ℓ, it assigns τ the 2ℓ-th priority value of all rows in sample set, and broadcasts it to all sites (line 3-4). The samples whose priority values are less than τ are moved to the candidate set S' (line 5-6). If the coordinator finds that $|S|$ is less than ℓ (due to expiration), it keeps halving the threshold, collecting valid rows (i.e., rows with priority values greater than τ) from S' and all sites, and inserting such rows into S, until $|S|$ exceeds 2ℓ (line 7-11). Besides, on acquiring a new threshold τ from the coordinator, site S_j compares it with the old τ_j. If $\tau < \tau_j$, site S_j should send all active rows with priority values greater than τ to the coordinator, and remove them from Q (line 13-16). Finally, site j sets local threshold $\tau_j = \tau$ (line 17).

Analysis. The correctness of the protocol is easy to prove, which we omit. The space usage is the same. We next analyze the communication cost and the number of threshold updates.

Lemma 3. *During any time interval $(t, t+W]$, the communication cost of the lazy-broadcast protocol is $O(d\ell \log(NR) + m \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$, and the number of updates of τ is $O(\log(NR))$ with high probability.*

Proof: With a slightly more complicated argument than that in the proof of Lemma 2, we can prove that all sites send totally $O(\ell \log(NR))$ rows per window with high prob-

ability (see full version [31]). Now we bound the number of broadcasts, i.e., the number of times τ is updated. Consider two consecutive increases of τ at t_1 and t_2 respectively. At time t_1+1, the size of S becomes 2ℓ, while at t_2 the size of S is 4ℓ. Note that we cannot simply conclude that the exact 2ℓ rows are sent from all sites, since the rows in S' could be added back to S. We consider two cases. If τ does not decrease between t_1 and t_2, then no rows in S' will be added back to S. Therefore, at least 2ℓ rows are sent from sites during this time period. Otherwise, if τ decreases at some time $t^* \in (t_1, t_2)$, then there are at least ℓ rows in S from time t_1 have expired. As a result, between any two consecutive increases of τ, either 2ℓ rows are sent to the coordinator or at least ℓ samples expire. During a time window, the total number of rows sent is $O(\ell \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$ and the number of sampled rows get expired is at most $O(\ell \log(NR))$ with probability $1 - e^{-\Omega(\ell)}$, and thus the number of times that τ increases is at most $O(\log(NR))$ with probability $1 - e^{-\Omega(\ell)}$.

Now consider the moment when $|S|$ becomes ℓ, i.e., τ needs to be halved. Assume β is the minimum weight among all rows, and thus the maximum weight is $R\beta$. We claim that, with high probability, $\tau \leq NR\beta$ (proof in the full version [31]). On the other hand, the minimum possible priority value is β, and thus the number of iterations of the while loop in line 8 of Algorithm 2 is at most $O(\log(NR))$. After this, $|S| \geq 2\ell$, which means before the next time when $|S| = \ell$, there are at least ℓ rows in S expiring. Therefore, the number of times that $|S|$ reaches ℓ is $O(\log(NR))$ during a time window (since $O(\ell \log(NR))$ sampled rows will expire during a time window with high probability, as shown in the proof of Lemma 2). Since we have shown, each time $|S|$ becomes ℓ, it incurs $O(\log(NR))$ updates of threshold, the threshold τ is updated for at most $O(\log^2(NR))$ times during a time window with high probability. This can be reduced to $O(\log(NR))$ by a more careful analysis. ■

Output a covariance sketch. It was known that (e.g. [1]), we only need to sample $\ell = O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ rows to achieve ε-covariance error with high probability. To get a valid covariance sketch, we rescale all sampled rows properly to get unbiased estimators. For priority sampling, we assign each sampled row a_i a new weight $v_i = \max\{\|a_i\|^2, \tau_\ell\}$ [26], where τ_ℓ is the ℓ-th largest priority value, i.e., set $a_i = \frac{v_i}{\|a_i\|^2} a_i$ for all sampled rows. By stacking all rescaled rows together, the matrix $B = [a_1; a_2; \cdots; a_\ell]$ is an ε-covariance sketch.

Note that in lazy-broadcast protocol the coordinator maintains a sample set with at least ℓ samples. Intuitively, the accuracy will be improved by using all available samples, which are actually "free". Based on this observation, we also implemented a variation of PWOR, denoted as PWOR-ALL, which makes use of all samples available to the coordinator. A comparison of PWOR and PWOR-ALL will be presented in the experimental section.

Priority Sampling with Replacement (PWR). We briefly discuss priority sampling with replacement. This can be done by maintaining ℓ independent samplers in the coordinator, each of which uses the above PWOR protocol to maintain $O(1)$ samples. Although the communication cost is similar to PWOR, this PWR protocol needs to maintain ℓ independent thresholds, which requires $O(\ell \log(NR))$ threshold synchronizations. To avoid the large cost of broadcasting thresholds, we can use a similar sharing-threshold method as in [2] to solve our problem. The details are included in the full version [31].

B. ES Sampling

In ES sampling without replacement (ESWOR), each row a_i is assigned a random priority $u_i^{1/\|a_i\|^2}$, where u_i is a random variable drawn uniformly from $(0, 1)$. To sample ℓ items without replacement, we also pick the ℓ items with highest priority values. Therefore, the protocol for ES sampling follows the same framework as for the priority sampling. With some minor modification, the analysis for priority sampling directly works for ES sampling. To get a covariance sketch, each sampled row also needs to be rescaled properly: each row a_i in the sample set is rescaled by a factor of $\sqrt{\ell}\|a_i\|/\|A_w\|_F$ [17]. One technical issue is we need to know $\|A_w\|_F^2$. Fortunately a $(1+\varepsilon)$ approximation to $\|A_w\|_F^2$ is good enough, which is equivalent to SUM tracking problem. Tracking sum can be solved either by priority sampling above, or by a deterministic method to be presented in Section III-A. Compared to the cost of row sampling, the additional cost to track $\|A_w\|_F^2$ by both methods is negligible. ES sampling with replacement (ESWR) follows the same framework as PWR.

III. Deterministic methods

We will introduce two deterministic methods for tracking covariance sketch. To illustrate the idea of our first method, we will first give a deterministic protocol for tracking sum over a weighted sliding window stream, which generalizes and simplifies a protocol in [28] for tracking count.

A. Deterministic SUM tracking

In the SUM tracking problem, each item in the stream has a weight w_i, the goal is to track an estimator to the sum of weights of all active items. The SUM tracking problem is a special case of matrix tracking (d=1). In addition, when all weights are 1, this becomes tracking COUNT, i.e., the total number of items, which was studied in [10, 28]. In particular, [28] proposed a general *forward-backward* framework, which solves the COUNT problem with optimal communication cost. Let $C^{(j)}(t)$ be the number of active items on site S_j at time t, then $C(t) = \sum_j^m C^{(j)}(t)$ is the exact answer to the COUNT problem. To track $C(t)$ within relative ε error, it is sufficient to track each $C^{(j)}(t)$ within ε error. The forward-backward framework [28] is for tracking each individual $C^{(j)}(t)$. Together with some rounding argument as in [19], the weighted version can also be solved by forward-backward tracking.

A more direct and efficient approach for tracking SUM. A more intuitive way is to track the difference between $C^{(j)}(t)$ and $\hat{C}^{(j)}(t)$, where $\hat{C}^{(j)}(t)$ is the current SUM estimator the coordinator holds (note the site S_j also knows $\hat{C}^{(j)}(t)$) and $C^{(j)}(t)$ is the actual SUM the coordinator wishes to track. For notational convenience, we omit the superscripts and simply write $C^{(j)}$ as C and $\hat{C}^{(j)}$ as $\hat{C}$. More precisely, S_j maintains $D = C - \hat{C}$, and whenever $|D| > \varepsilon C$, the site sends D to the coordinator and updates the estimator $\hat{C} = C$. The correctness is straightforward, since at any time t, $|C - \hat{C}| \leq \varepsilon C$. The naive way to maintain exact C (or D) needs $O(N)$ space. However, maintaining C exactly is not necessary, and we only

Algorithm 3 Improved Protocol for Tracking SUM

```
1: procedure TRACK_SUM(time t)                    ▷ at site j
2:     for (w_i, t_i) = receive_data(t) do
3:         Insert (w_i, t_i) into gEH;
4:     C = gEH.query().
5:     if |C − Ĉ| > εC then
6:         send D = C − Ĉ;
7:         Ĉ = C.
8: procedure RECEIVE_UPDATE(D)                    ▷ at coordinator
9:     Ĉ = Ĉ + D.
```

need to track an ε-approximation C' to C, which can be done in $O(\frac{1}{\varepsilon}\log(WR))$ space by using *generalized exponential histogram* (gEH) [19]. Note that using ε-approximation C' instead of the exact sum C does not affect the correctness: by adjusting ε by a constant factor in the beginning, we still promise an ε relative error for tracking SUM.

Theorem 1. Algorithm 3 solves the SUM tracking problem over distributed sliding windows, which incurs $O(\frac{m}{\varepsilon}\log(NR))$ words of communication per window. The space usage is $O(\frac{1}{\varepsilon}\log(NR))$ words per site.

Proof: The correctness has been proved above, so we analyze the number of messages sent by a single site j. We divide such messages into two types: positive updates ($D > 0$) and negative updates ($D < 0$). For simplicity we assume C is exact (i.e., without using gEH). The case where C is only an ε relative approximation can be analyzed similarly with more care. Let us consider any fixed time interval $(t-2W, t]$. We call time window $(t-2W, t-W]$ the *left window*, and call $(t-W, t]$ the *right window*. Let $C_l(t_{now})$ be the sum of active items in the left window at time t_{now} ($t-W < t_{now} \le t$), i.e., the sum of items in the window $(t_{now}-W, t]$, and $C_r(t_{now})$ be the sum of active items in the right window, i.e., the sum of items in the window $(t-W, t_{now}]$. We first bound the number of positive updates. Consider any two consecutive positive updates that happen at t_1 and t_2 respectively. By definition, $C(t_2) - C(t_1) > \varepsilon C(t_2)$, where $C(t)$ denotes the value of the variable C at time t in the algorithm. This means the sum of weights of items arrive in the time interval $(t_1, t_2]$ is at least $\varepsilon C(t_2)$, i.e., C_r increases by $\varepsilon C(t_2)$. Clearly $\varepsilon C(t_2) \ge \varepsilon C_r(t_2)$, and thus C_r increases by a factor of $1+\varepsilon$ during any two consecutive positive updates. Since C_r is bounded by NR, the number of positive updates is at most $\log_{(1+\varepsilon)}(NR) = O(\frac{1}{\varepsilon}\log(NR))$. The number of negative updates can be bounded similarly. The space used by each site is dominated by the gEH, which is $O(\frac{1}{\varepsilon}\log(NR))$. Since query and update time of gEH is amortized $O(1)$ [19], the update time of the algorithm is amortized $O(1)$. ■

B. First deterministic protocols for tracking covariance sketch

Let $A_w^{(j)}$ be the matrix consisting of active rows at site S_j, so $A_w = [A_w^{(1)}; \cdots; A_w^{(m)}]$ is the target matrix. If we can track each $A_w^{(j)}$ separately within covariance error ε, i.e., the coordinator has a matrix $B^{(j)}$ for each site S_j such that $\|A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\| \le \varepsilon\|A_w^{(j)}\|_F^2$, then matrix $B = [B^{(1)}; \cdots; B^{(m)}]$ is a ε covariance sketch with respect to A_w. To see that, we have

$$\begin{aligned}\|A_w^TA_w - B^TB\| &= \|\sum_{j=1}^{m}\left(A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\right)\| \\ &\le \sum_{j=1}^{m}\|A_w^{(j)T}A_w^{(j)} - B^{(j)T}B^{(j)}\| \\ &\le \sum_{j=1}^{m}\varepsilon\|A_w^{(j)}\|_F^2 = \varepsilon\|A_w\|_F^2,\end{aligned}$$

where we use triangle inequality for spectral norm in the first inequality above. Therefore, we only consider the problem of tracking the sliding window matrix on each site separately, and for notational convenience, we omit the superscripts and simply write $A_w^{(j)}$ as A_w at below (but it actually means the sliding window matrix at a single site).

High level idea of DA1. First, assume each site is allowed to store all rows received in the current time window. We use the same "template" as in SUM tracking, except now C and $\hat{C}$ are $d \times d$ matrices. Let $C = A_w^TA_w$ and $\hat{C} = B^TB$, where B is the covariance sketch of A_w maintained by the coordinator. Site S_j maintains C and $D = C - \hat{C}$ exactly, and whenever $\|D\| \ge \varepsilon\|A_w\|_F^2$, S_j reports the variation. The key difference here is that S_j does not send the whole D, but sends the "significant directions" of D only, so that the remaining difference is tolerable. Otherwise, the communication cost is unbounded. Let $D = \sum_{i=1}^{d}\lambda_i v_i^Tv_i$ be the standard eigen-decomposition of D, i.e., each v_i is an eigenvector (as a row vector) of D corresponding to the eigenvalue λ_i. The site S_j sends (λ_i, v_i) for all i such that $|\lambda_i| > \varepsilon\|A_w\|_F^2$, and updates $\hat{C} = \hat{C} + \lambda \cdot v_i^Tv_i$. For each (λ_i, v_i) being received, the coordinator also updates its $\hat{C}$ accordingly. Now we have $\|A_w^TA_w - \hat{C}\| = \|C - \hat{C}\| \le \varepsilon\|A_w\|_F^2$ on the coordinator. To get a ε-covariance sketch to A_w, we compute the *matrix square root* B of $\hat{C}$, i.e., $\hat{C} = B^TB$ [32].[2] Maintaining C (or D) exactly needs to store all active rows. As in the SUM tracking protocol, here we will use the *matrix exponential histogram* (mEH) of [17] to save space. The mEH takes $O(\frac{d}{\varepsilon^2}\log(NR))$ words of space, and always maintains an ε-covariance sketch C' to $A_w^TA_w$. Besides, mEH maintains an ε relative estimate $\|\hat{A}_w\|_F^2$ to $\|A_w\|_F^2$. By adjusting ε by a constant factor in the beginning, we still promise an ε covariance error with our approach. For simplicity, we use C instead of C' in all illustrations. The pseudocode of DA1 is shown in Algorithm 4. By a similar (but more technical) analysis as in the SUM tracking protocol, we can show that the communication cost per window is $O(\frac{md}{\varepsilon}\log(NR))$ (see [31]). The space usage is dominated by mEH and $\hat{C}$, which is $O(\frac{d}{\varepsilon^2}\log(NR) + d^2)$.

Limitations of DA1. However, there are a few limitations to the above deterministic algorithm. First, we need to compute eigen-decomposition of a $d \times d$ matrix (i.e., D) frequently, each of which takes $O(d^3)$ time. Secondly, even with matrix exponential histogram, it requires additional d^2 space and $O(d^2)$ time per update to maintain D. Hence, the algorithm is not suitable for large d. Motivated by these, we propose a second deterministic algorithm DA2 with better scalability in terms of d. Empirical comparisons of two deterministic algorithms will be shown in the experiments section.

[2] B exists, since $\hat{C}$ on the coordinator is *positive semidefinite*, which can be computed with SVD.

Algorithm 4 DA1 for Tracking Matrix Approximation

procedure TRACK_MATRIX(time t) ▷ at site j
  for row (a_i, t_i) = $receive\_row(t)$ **do**
    Insert (a_i, t_i) into mEH;
  $(C, \|\hat{A}_w\|_F^2) = mEH.query()$;
  $D = C - \hat{C}$.
  if $\|D\| > \varepsilon\|\hat{A}_w\|_F^2$ **then**
    Compute $D = \sum_{i=1}^{d} \lambda_i v_i^T v_i$. ▷ Compute eigen-decomposition of D
    for $|\lambda_i| \geq \varepsilon\|\hat{A}_w\|_F^2$ **do**
      $\hat{C} = \hat{C} + \lambda \cdot v_i^T v_i$;
      Send (λ_i, v_i).
procedure RECEIVE_UPDATE(λ, v_i) ▷ at coordinator
  $\hat{C} = \hat{C} + \lambda \cdot v_i^T v_i$.
procedure QUERY() ▷ at coordinator
  $(U, \Sigma, V) = \mathsf{SVD}(\hat{C})$;
  return $\Sigma^{1/2} V^T$

High level idea of DA2. The most time-consuming part of DA1 is to compute the eigen-decomposition of D. In DA2, we will use the *frequent direction* (FD) algorithm of [13] to dynamically maintain an approximate of D. The challenge is that D takes negative updates, i.e., expiration of old rows, while FD only handles positive updates. The idea of DA2 is motivated by the forward-backward framework [28]. The whole tracking period is divided into windows of length W: $(0, W], (W, 2W], \cdots, (iW, (i+1)W], \cdots$. Assume the current time is $kW < t_{now} < (k+1)W$, the target matrix is a concatenation of two sub-matrices, namely $A_e(t_{now})$ and $A_a(t_{now})$, corresponding to the *expiring window* $(t_{now} - W, kW]$ and *active window* $(kW, t_{now}]$ respectively. When there is no confusion, we simply write them as A_e and A_a. As the time window $(t_{now} - W, t_{now}]$ slides, new rows are added to A_a while old rows expire from A_e. We then track A_a and A_e separately with ε covariance error. Tracking A_a is essentially the same as the infinite window case, which can be solved by a protocol from [1] combined with FD (for efficiency). However, instead of tracking A_e directly, we track $M_e(t_{now})$, the matrix in $((k-1)W, t_{now} - W]$. Since we have $A_e(kW) = [M_e(t_{now}); A_e(t_{now})]$ for any $t_{now} \in (kW, (k+1)W]$, and an ε-covariance sketch of $A_e(kW)$ is already known by coordinator at time kW, thus given an ε-covariance sketch to $M_e(t_{now})$, the coordinator can compute an ε-covariance sketch to $A_e(t_{now})$. In practice, instead of tracking A_e and A_a separatively, the coordinator maintains $\hat{C} = B^T B$ as in DA1. More precisely, the coordinator simply updates the newly observed directions (positive) and expired directions (negative) on $\hat{C}$, but doesn't have to operate eigen decomposition on $\hat{C}$, thus the overall performance won't be affected.

In the following part, we first assume each site stores all active rows. Similar as in DA1, we can use matrix exponential histogram to save space. Although the original infinite window matrix tracking protocol (P2) in [1] needs two-way communication, here we only use this for tracking a single site, so it is essentially a one-way protocol. We call this IWMT protocol. We regard IWMT as a "black box", which receives as input a sequence of rows and sends/outputs another row sequence, which actually consists of the "significant directions" with the corresponding timestamps; the covariance error between two matrices formed by any prefixes (w.r.t. a specific timestamp) of the two row sequences is bounded by a given threshold.

We use a IWMT protocol to track A_a (forward tracking), namely $IWMT_a$. On arrival of new row (a_i, t_i), site S_j inputs it into $IWMT_a$ and sends the out message m_i, where m_i is a row and t_i is the timestamp. The coordinator adds $m_i^T m_i$ to $\hat{C}$, an ε-covariance sketch of $A_w^T A_w$. Tracking A_e (backward tracking) is more complicated. At t_{now}=kW for some integer k, site S_j runs a IWMT protocol, namely $IWMT_c$, reading the row sequence of $A_e(kW)$ in the reverse time direction locally, and uses a queue Q to record every message (m_i, t_i) from $IWMT_c$[3]. Recall that we actually want to track $M_e(t)$. To do so, site S_j starts another IWMT protocol at $t = kW$, namely $IWMT_e$, which does a forward tracking on Q according to expiring time, i.e., at any time, we feed $IWMT_e$ each expired row m_i in Q and send the output. More formally, let $Q = \{(m_i, t_i) \mid i = 1, 2, \cdots, \}$ be the messages recorded by applying $IWMT_c$ on $A_e(kW)$ in the reverse order of time. Since site S_j has $A_e(kW)$ at time kW, Q can be computed instantly by simulating IWMT on the row stream of $A_e(kW)$ in the reverse order. Then, during $t_{now} \in (kW, (k+1)W]$, $IWMT_e$ reads the expired rows from Q as t_{now} increases, and sends the output messages if necessary.

Note in IWMT protocol [1], a threshold is used to limit the maximum norm of unsent content. Basically, the larger the threshold is, the less number of messages are sent. The thresholds for three IWMT protocols in our algorithm are set as follow: (1) the input of $IWMT_c$ is a row sequence of descending timestamp (in a reverse order), and on receiving (a_i, t_i), $IWMT_c$ uses $\varepsilon\|\hat{A}_e(t_i + W)\|_F^2$ as threshold, i.e., the total squared norms of rows received by $IWMT_c$ thus far. This follows exactly the original IWMT protocol. (2) both $IWMT_a$ and $IWMT_e$ use $\varepsilon\|\hat{A}_w\|_F^2$ as the threshold during $t \in (kW, (k+1)W]$. This is actually an *optimal* threshold, thus the overall communication cost of DA2 can potentially be much lower compared to a basic forward-backward protocol.

The worst-case communication cost during the time window $(kW, (k+1)W]$ is no more than twice the cost of IWMT for tracking a single site, which is $O(\frac{d}{\varepsilon}\log(NR))$ [1], and thus the total communication cost is $O(\frac{md}{\varepsilon}\log(NR))$ per window. The amortized update time of the IWMT protocol is $O(\frac{d}{\varepsilon})$ if we use frequent direction of [13] to speed up the computation. The space usage is dominated by a matrix exponential histogram used, which is $O(\frac{d}{\varepsilon^2}\log(NR))$ [17].

IV. EXPERIMENTS

A. Experiments Setting

In our experiments, we compare all proposed algorithms on data matrices with various characteristics, aiming to provide useful guidance on using these algorithms as building blocks in various applications. Based on our theoretical analysis, we expect that different algorithms would perform better under different settings of parameters. A detailed analysis of the experimental results will be further provided.

[3] $IWMT_c$ process A_e in the reverse order, but the timestamp of each message is recorded using the actual time, and thus the messages in Q is ordered in the reverse order of time.

Algorithm 5 DA2 for Tracking Matrix Approximation

procedure TRACK_MATRIX(time t_{now}) ▷ at site j
  for row a_i = $receive\_row(t_{now})$ **do**
    $m_i = IWMT_a(a_i, t_{now})$; Send $(m_i, flag = 1)$;
    Insert (a_i, t_{now}) into mEH_a;
  if $t_{now} == kW$ for integer k **then** ▷ End of window
    $Q = IWMT_c(mEH_a)$.
  for $(m_i, t_i) \in Q$ and $t_i < t_{now} - W$ **do**
    $m_i' = IWMT_e(m_i, t_i)$; Send $(m_i', flag = -1)$;
    Remove (m_i, t_i) from Q.
procedure RECEIVE_UPDATE(m_i,$flag$) ▷ at coordinator
  $\hat{C} = \hat{C} + flag \times m_i^T m_i$.
procedure QUERY() ▷ at coordinator
  $(U, \Sigma, V) = \mathsf{SVD}(\hat{C})$;
  return $\Sigma^{1/2} V^T$

Datasets. We use two publicly available datasets "PAMAP" and "WIKI", as well as a synthetic data set in the experiments. The data sets are summarised in Table III.

PAMAP[4] is a Physical Activity Monitoring dataset, which is the data of 18 different physical activities (such as walking, cycling, playing soccer, etc.), performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor. The dataset contains 54 columns including a timestamp, an activity label and 52 attributes of raw sensory data. In our experiments, we used a subset with $N = 814729$ rows and $d = 43$ columns (removing columns containing missing values). We generate synthetic timestamps for PAMAP following the *Poisson Arrival Process* [33] with $\lambda = 1$. Moreover, we set the window size such that on average there are approximately $200,000$ rows in a window.

WIKI[5]is the text corpus built on the article dump of the September 2015 version of English Wikipedia. The words occurring at least 1000 times in the entire corpus are used as features (columns), and the articles with at least 500 features are selected as rows. The entry at row i and column j is the $tf\text{-}idf$ weight of article i and word j. The timestamp of each row is the date when the article is published, i.e., we view one day as a time unit. The matrix consists of 78608 rows and 7047 columns. The timestamp spans over 3949 days. We set the window size to 502 such that on average there are 10000 rows in a window.

SYNTHETIC is a random noisy matrix that have been commonly used for evaluating matrix sketching algorithms [17][1][13]. The matrix is concatenated by 3 submatrices of the same size. Each submatrix is generated by formula $A{=}SDU{+}N/\zeta$, where S is a $n \times d$ coefficients matrix with each entry drawn from standard normal distribution, D is a diagonal matrix with $D_{i,i}{=}1{-}(i{-}1)/d$, U is a random matrix satisfying $UU^T{=}I_d$, N contributes additive Gaussian noise with each entry drawn from standard norm distribution, and ζ is assigned 10 thus the real signal is recoverable. The matrix has 300 columns and $500,000$ rows. We generate synthetic timestamps for SYNTHETIC, following the *Poisson Arrival Process* [33] with $\lambda = 1$. We set the window size such that on average there are approximately $100,000$ rows in a window.

[4]http://www.pamap.org/demo.html

[5]https://en.wikipedia.org/wiki/Wikipedia:Database_download

Data Sets	total rows n	d	average rows per window	ratio R
PAMAP	814,729	43	$\approx 200{,}000$	60.78
SYNTHETIC	500,000	300	$\approx 100{,}000$	3.72
WIKI	78,608	7047	$\approx 10{,}000$	2998.83

Table III: Summary of Data sets.

Algorithms. We compared all of the protocols proposed in this paper, including sampling based and deterministic tracking algorithms. Since there is no previous work for matrix sketching that can be adapted to the distributed sliding window model, we view random sampling as the baseline approach.

(1) Priority sampling. We evaluated priority sampling without replacement PWOR, and its variation PWOR-ALL which makes use of all "candidate samples". Intuitively PWOR-ALL has better accuracy than PWOR, while we still present the performance of PWOR in our experiment results, since some applications are required to output exactly ℓ samples, which is the standard definition of random sampling with or without replacement. Moreover, it is observed that using more available samples does not always improve accuracy (details will be shown later), which is interesting for comparison.

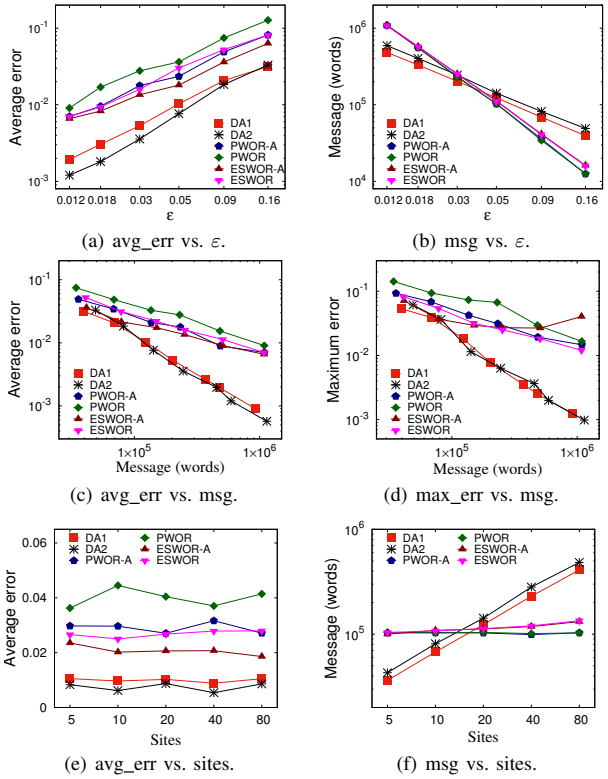
(2) ES sampling. We evaluated ESWOR, the ES sampling algorithm for without-replacement scheme, and ESWOR-ALL, a variation of ESWOR that makes use of all "candidate samples". ES sampling differs from priority sampling by using different priority functions and estimators, hence it is interesting to compare their performances on real world datasets.

(3) Deterministic. We evaluated the DA1 and DA2 algorithms. We remark that DA1 is very slow for large dimension d, and could not finish on WIKI, so we only present the results of DA1 on PAMAP and SYTHETIC.

Note that we excluded the with-replacement sampling algorithms from our experiments. Firstly, sampling with replacement is extremely time-consuming, and thus is infeasible for large datasets. Secondly, as observed in [1] and [17], SWOR is at least as accurate as SWR on most datasets.

Metrics. In the experimental study, we used default value $\varepsilon = 0.05$ and $m = 20$. Note that for all of our protocols, the theoretical analyses only show asymptotic worst-case communication costs, which rarely happen in real data sets. To evaluate above algorithms, we tested the trade-off between actual communication cost and observed covariance error. We also measured the communication cost and accuracy as ε and m varied, as well as the update rate and space usage of each site. The metrics are defined as follows: 1) Communication cost *msg* is defined to be the average number of words sent per window. We assume each real number takes 1 word. 2) Approximation error *err* is defined to be $\|A_w^T A_w - B^T B\|_2 / \|A_w\|_F^2$, where A_w is the matrix of current sliding window and B is the matrix approximation. We randomly picked 50 query points and measured both average and maximum error (denoted by avg_err and max_err).

Setup. We simulated the distributed system on a single machine to evaluate the proposed metrics, which is a standard methodology adopted in the distributed monitoring literature. We agree that implementing distributed monitoring algorithms in a real distributed system and evaluating the overall performance will be more instructive, which is left as our future work. For random sampling, we ran the algorithms for 3 times in each experiment, and reported the average communication

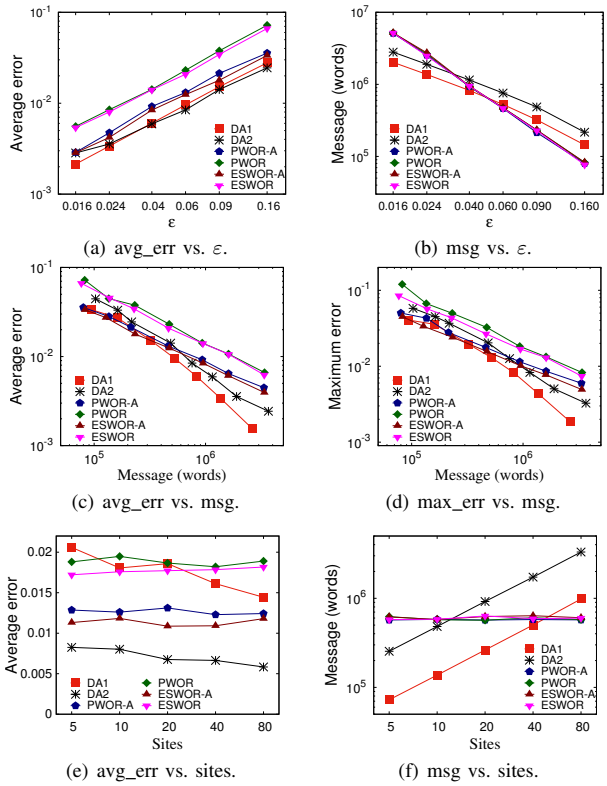


(a) avg_err vs. ε. (b) msg vs. ε.

(c) avg_err vs. msg. (d) max_err vs. msg.

(e) avg_err vs. sites. (f) msg vs. sites.

Figure 1: Results on PAMAP dataset.



(a) avg_err vs. ε. (b) msg vs. ε.

(c) avg_err vs. msg. (d) max_err vs. msg.

(e) avg_err vs. sites. (f) msg vs. sites.

Figure 2: Results on SYNTHETIC dataset.

cost and error over 3 executions. All algorithms were implemented in Python 3.5.1 using 64-bit addressing and were executed on Intel Xeon E3-1231V3, clocked at 3.4 GHz.

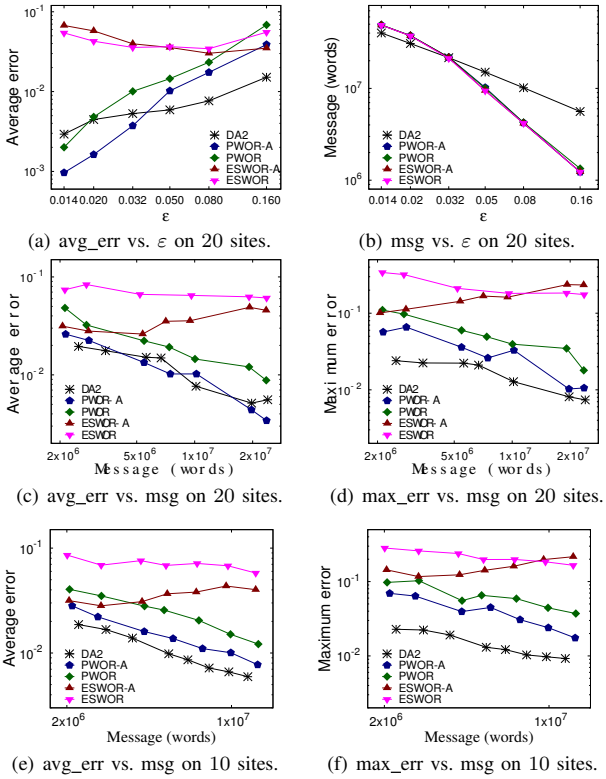
B. Performance Study

Error vs. Communication cost. We make the following observations:

(1) Figure 1(a), 2(a), and 3(a) show the tradeoffs between the error parameter ε and the average actual error of the protocols. In most cases, the observed error for all protocols is smaller than ε. We also observe that deterministic protocols provides better error guarantee than sampling methods do, under the same error parameter ε. Inside the sampling family, using all available samples typically achieves better accuracy.

(2) Figure 1(b), 2(b), and 3(b) show the tradeoffs between the error parameter ε and the communication cost of protocols. It is worth noticing that as ε decreases, the communication cost of deterministic protocols grows much slower than random sampling does, which confirms the dependency of their asymptotic bound on ε ($1/\varepsilon$ vs. $1/\varepsilon^2$). Communication cost of ES sampling is slightly higher than that of priority sampling, which results from the extra cost of tracking $\|A_w\|_F^2$.

(3) The tradeoffs between observed error and communication cost are shown in Figure 1(c),1(d), Figure 2(c),2(d), and Figure 3(c),3(d) respectively. In the default setting where $m = 20$, DA1 and DA2 achieve better "communication vs error" tradeoff than that of the sampling methods. The advantage of the deterministic algorithms is more significant for maximum error. This is as expected, since the error guarantee



(a) avg_err vs. ε on 20 sites. (b) msg vs. ε on 20 sites.

(c) avg_err vs. msg on 20 sites. (d) max_err vs. msg on 20 sites.

(e) avg_err vs. msg on 10 sites. (f) max_err vs. msg on 10 sites.

Figure 3: Results on WIKI dataset.

of sampling methods is randomized. For the two deterministic protocols, we have the following observations: (i) DA1 is notably better on SYNTHETIC dataset, where rows of the matrix are generated from the same distribution; (ii) The two protocols have similar performance on PAMAP; (iii) DA1 is too slow to finish the experiments on WIKI. For sampling methods, making use of all available samples (PWOR-ALL and ESWOR-ALL) usually achieves better tradeoff than top-ℓ samples (PWOR and ESWOR). Furthermore, PWOR-ALL significantly outperforms ESWOR-ALL on WIKI dataset, while ESWOR-ALL has slight advantage on PAMAP dataset. They both behave well on SYNTHETIC dataset.

(4) Another interesting observation is that, on skewed datasets (PAMAP and WIKI) (i.e. R is large), increasing the communication does not necessarily reduce the error of ESWOR-ALL (Figure 1(d), 3(c) and 3(d)). One possible explanation is that ES sampling rescales each sample a_i by a factor of $\frac{\|A_w\|_F}{\sqrt{\ell'}\|a_i\|}$, after which all samples have the same squared norm. As a result, a sampled row with small norm will be greatly stretched, and is over-emphasized in final estimation. When the size of sample set increases, such "bad" events occur more frequently and lead to relative high covariance error. This conjecture is also supported by Figure 3(c) and 3(d), which show that the maximum error of ESWOR-ALL is higher than that of ESWOR when communication cost exceeds 10^7 words. We believe that this is because ESWOR only takes top-ℓ samples, and thus decreases the risk of picking rows with small norms. On the other hand, priority sampling rescales a_i to make its squared norm equal to $\max\{\|a_i\|^2, \tau_\ell\}$, which sets a *ceiling* τ_ℓ for rows with small norms while ensuring the contribution of rows with large norms. Hence we recommend priority sampling rather than ES sampling on skewed datasets.

Varying number of sites. We also conducted experiments to show how the error and communication cost changes as the number of sites varies. We set the number of sites for PAMAP and SYNTHETIC to vary from 5 to 80. The total number of rows is relatively small for WIKI, so we only tested the performance for 10 and 20 sites to make sure that each site receives enough rows. We make the following observations: (1) The covariance error of all protocols is stable as m varies (Figure 1(e) and 2(e)); (2) For sampling methods, the communication cost remains the same as m increases (Figure 1(f) and 2(f)). This is consistent with our analysis, since the communication cost is dominated by the term $O(\frac{d}{\varepsilon^2}\log(NR))$, which does not depend on m. (3) The communication cost of deterministic protocols clearly has a linear dependence on the number of sites, which matches our theoretical analysis (both DA1 and DA2 need $O(\frac{md}{\varepsilon}\log(NR))$ words of communication). In general, the sampling protocols are scalable with the site number while the deterministic algorithms are not.

Space usage and update rate. Finally, we measured the space usage and update rate of all protocols. Figures 4(a)4(b)4(c) shows the tradeoffs between maximum space usages and the error parameter ε of all protocols. When ε is small, all protocols store almost all rows observed in the window. In most cases, the space usages of all protocols are close to each other, which confirms our theoretical analysis: the space needed by each site is roughly $\widetilde{O}(\frac{d}{\varepsilon^2})$ words for both deterministic and sampling protocols. DA1 requires extra $d\times d$ space to maintain matrix sketches, which is negligible when d is small (PAMAP and SYNTHETIC). On the WIKI dataset, however, the space usage of DA2 does not decrease significantly as ε increases. The reason is that the large ratio R (2998.83 to be exact) of WIKI dataset greatly limits the compression effect of the mEH structure. We also note that the space usage of sampling methods is small when ε is set to be small on WIKI dataset. This is because that each site sends many rows to coordinator for the large sample set, and thus the number of rows left in each site is greatly decreased.

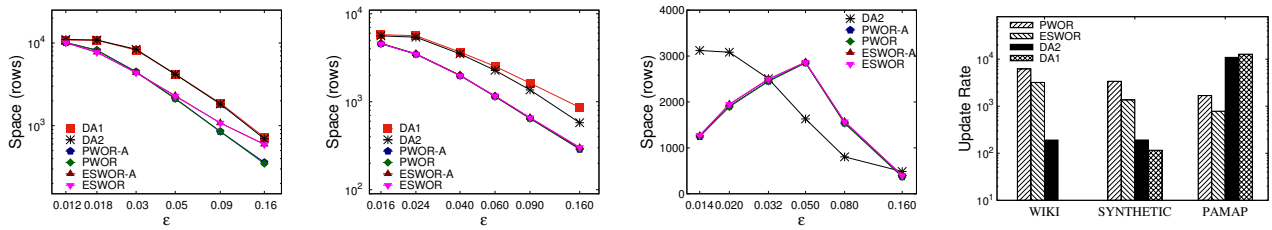
Figure 4(d) shows the update rates of protocols (updates processed per second) under the default setting where $\varepsilon = 0.05$ and $m = 20$. We observe that deterministic protocols process updates faster than sampling methods for low-dimensional matrix (PAMAP), while their update rate decreases dramatically as d increases. This is because DA1 and DA2 need to compute matrix factorizations periodically, which leads to running time quadratic or even cubic in d. For WIKI dataset (where $d \approx 7000$), DA1 is too slow to finish the experiments, while the update rate for DA2 is 10 to 100 times slower than sampling methods. Notice that the update rate of sampling methods is not affected by d. Interestingly, the sampling algorithms process update faster on WIKI than on PAMAP. This is reasonable, since d only affects the time to compute the norm, which is not the dominating part of the running time.

C. Remarks

We conclude our experimental evaluations with a few remarks. In general, the deterministic protocols achieves better "communication vs. error" tradeoff than sampling methods. Moreover, the deterministic protocols achieve deterministic error guarantee, which is desirable in some applications. So if high accuracy is the main concern, the deterministic protocols are recommended. DA1 and DA2 have similar performance in terms of accuracy and communication, while DA2 outperforms DA1 in processing time. Therefore, we recommend DA2 for large d for its time-efficiency, and recommend DA1 when d is small as it is much easier to implement. On the other hand, the sampling methods trade accuracy for some other desirable properties. For instance, sampling methods produce a sketch consists of rows of the original matrix, which improves interpretability and preserves row structure. This is essential for some applications such as *column/row subset selection* [34]. Moreover, the processing time for the sampling methods process is much faster than that of the deterministic algorithms when d is large. Inside the sampling schemes, the overall performance of PWOR-ALL is the best, and thus we recommend PWOR-ALL for matrix approximation.

V. Conclusion

In this paper, we initialize the study of continuously tracking a matrix approximation over distributed sliding windows, and propose protocols with low communication cost and space usage for the problem. We first consider row sampling methods, and raise novel algorithms for tracking weighted random samples over distributed sliding windows, which generalize and simplify previous works for unweighted sampling. We also provide two deterministic protocols which only use one-way communication and have a better dependence on ε in terms of communication cost. In addition to theoretical analysis, extensive experiments are conducted on large-scale



(a) space vs. ε on PAMAP. (b) space vs. ε on SYNTHETIC. (c) space vs. ε on WIKI. (d) Update Rate.

Figure 4: Space usage and update rate vs. ε on all datasets.

real and synthetic data sets, and a detailed comparison of their performances is provided.

REFERENCES

[1] M. Ghashami, J. M. Phillips, and F. Li, "Continuous matrix approximation on distributed data," *Proceedings of the VLDB Endowment*, 2014.

[2] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Continuous sampling from distributed streams," *Journal of the ACM (JACM)*, vol. 59, no. 2, p. 10, 2012.

[3] G. Cormode, "The continuous distributed monitoring model," *ACM SIGMOD Record*, 2013.

[4] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *ACM TODS*, vol. 32, no. 4, p. 23, 2007.

[5] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," *TALG*, 2011.

[6] Z. Huang, K. Yi, and Q. Zhang, "Randomized algorithms for tracking distributed count, frequencies, and ranks," in *Proceedings of PODS*, 2012.

[7] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster, "Prediction-based geometric monitoring over distributed data streams," in *SIGMOD*, 2012.

[8] M. B. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proceedings of PODS*, 2004.

[9] A. Manjhi, S. Nath, and P. B. Gibbons, "Tributaries and deltas: Efficient and robust aggregation in sensor network streams," in *SIGMOD*, 2005.

[10] H.-L. Chan, T.-W. Lam, L.-K. Lee, and H.-F. Ting, "Continuous monitoring of distributed data streams over a time-based sliding window," *Algorithmica*, 2012.

[11] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM Computer Communication Review*. ACM, 2004.

[12] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *ACM SIGMETRICS Performance evaluation review*, vol. 32, no. 1, 2004, pp. 61–72.

[13] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of SIGKDD*, 2013.

[14] M. Ghashami and J. M. Phillips, "Relative errors for deterministic low-rank matrix approximations," in *Proceedings of SODA*, 2014.

[15] H. Huang and S. P. Kasiviswanathan, "Streaming anomaly detection using randomized matrix sketching," *Proceedings of the VLDB Endowment*, 2015.

[16] M. Ghashami, E. Liberty, J. Phillips, and D. Woodruff, "Frequent directions: Simple and deterministic matrix sketching," *SICOMP*, 2016.

[17] Z. Wei, X. Liu, F. Li, S. Shang, X. Du, and J. Wen, "Matrix sketching over sliding windows," in *SIGMOD*, 2016.

[18] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall, "A wakeup call for internet monitoring systems: The case for distributed triggers," in *HotNets-III*, 2004.

[19] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SICOMP*, vol. 31, no. 6, pp. 1794–1813, 2002.

[20] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *SODA*, 2002.

[21] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of PODS*, 2002.

[22] Z. Karnin and E. Liberty, "Online pca with spectral bounds," in *Proceedings of COLT*, 2015, pp. 505–509.

[23] S. Yoo, H. Huang, and S. P. Kasiviswanathan, "Streaming spectral clustering," in *ICDE*, 2016.

[24] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *SIGKDD*, 2015.

[25] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, 1985.

[26] N. Duffield, C. Lund, and M. Thorup, "Priority sampling for estimation of arbitrary subset sums," *Journal of the ACM (JACM)*, vol. 54, no. 6, p. 32, 2007.

[27] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *IPL*, 2006.

[28] G. Cormode and K. Yi, "Tracking distributed aggregates over time-based sliding windows," in *SSDBM*, 2012.

[29] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *arXiv preprint arXiv:1411.4357*, 2014.

[30] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, "Sketch-based querying of distributed sliding-window data streams," *VLDB Endowment*, 2012.

[31] H. Zhang, Z. Huang, Z. Wei, W. Zhang, and L. Xuemin, "Tracking matrix approximation over distributed sliding windows." [Online]. Available: http://www.cse.ust.hk/~huangzf/TrackingFull.pdf

[32] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[33] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Poisson_point_process

[34] J. A. Tropp, "Column subset selection, matrix factorization, and eigenvalue optimization," in *SODA*, 2009.